International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

G.191

(03/2010)

SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

International telephone connections and circuits – Software tools for transmission systems

Software tools for speech and audio coding standardization

Recommendation ITU-T G.191

ITU-T

International Telecommunication Union

ITU-T G-SERIES RECOMMENDATIONS

TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

INTERNATIONAL TELEPHONE CONNECTIONS AND CIRCUITS	G.100–G.199
Transmission planning and the E-model	G.100–G.109
General Recommendations on the transmission quality for an entire international telephone connection	G.110–G.119
General characteristics of national systems forming part of international connections	G.120–G.129
General characteristics of the 4-wire chain formed by the international circuits and national extension circuits	G.130–G.139
General characteristics of the 4-wire chain of international circuits; international transit	G.140–G.149
General characteristics of international telephone circuits and national extension circuits	G.150–G.159
Apparatus associated with long-distance telephone circuits	G.160–G.169
Transmission plan aspects of special circuits and connections using the international telephone connection network	G.170–G.179
Protection and restoration of transmission systems	G.180–G.189
Software tools for transmission systems	**G.190–G.199**
GENERAL CHARACTERISTICS COMMON TO ALL ANALOGUE CARRIER-TRANSMISSION SYSTEMS	G.200–G.299
INDIVIDUAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON METALLIC LINES	G.300–G.399
GENERAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON RADIO-RELAY OR SATELLITE LINKS AND INTERCONNECTION WITH METALLIC LINES	G.400–G.449
COORDINATION OF RADIOTELEPHONY AND LINE TELEPHONY	G.450–G.499
TRANSMISSION MEDIA AND OPTICAL SYSTEMS CHARACTERISTICS	G.600–G.699
DIGITAL TERMINAL EQUIPMENTS	G.700–G.799
DIGITAL NETWORKS	G.800–G.899
DIGITAL SECTIONS AND DIGITAL LINE SYSTEM	G.900–G.999
MULTIMEDIA QUALITY OF SERVICE AND PERFORMANCE – GENERIC AND USER-RELATED ASPECTS	G.1000–G.1999
TRANSMISSION MEDIA CHARACTERISTICS	G.6000–G.6999
DATA OVER TRANSPORT – GENERIC ASPECTS	G.7000–G.7999
PACKET OVER TRANSPORT ASPECTS	G.8000–G.8999
ACCESS NETWORKS	G.9000–G.9999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T G.191

Software tools for speech and audio coding standardization

Summary

This revision only introduces changes to ITU-T G.191 Annex A, which describes the ITU-T Software Tools (STL) containing a high-quality, portable C code library for speech processing applications. This new release of the STL – dubbed STL2009 – incorporates many changes in software and manual. The new sets of tools added are: ITU-T G.728 fixed-/floating-point executables, basic ITU-T G.722 frame-erasure concealments (G.722 PLC0 and PLC1), program ROM counting, floating point complexity evaluation, stereo operation and bitstream truncation. There have been also updates such as addition of new filters to rate change tools, more options added for frequency response calculations, adaptation of error-insertion device to embedded-variable bitstream, addition of impulse responses to the reverberation processing tool and update to make ITU-T G.722/ITU-T G.192 compliant. Corrections of several known problems have also been implemented. The associated STL user's manual has also been updated.

This Recommendation includes an electronic attachment containing STL2009 Software Tool Library and manual.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T G.191	1993-03-12	XV
2.0	ITU-T G.191	1996-11-11	15
3.0	ITU-T G.191	2000-11-17	16
4.0	ITU-T G.191	2005-09-13	16
5.0	ITU-T G.191	2010-03-29	16

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Electronic attachment: STL2009 Software Tool Library and manual

Recommendation ITU-T G.191

Software tools for speech and audio coding standardization

1 General

In the process of generating speech and audio coding standards, the following situations often happen:

a) In many cases, experimental results generated with different software tools may not be directly compared.

b) Software tools used by different organizations may not perfectly conform to related ITU-T Recommendations, which may delay ITU-T standardization processes.

c) ITU-T Recommendations may leave scope for different implementations.

d) New speech and audio coding standards are increasing in complexity, leading to non-bitexact specifications; furthermore, appropriate testing procedures to assure interoperability of different implementations are needed.

The need for a common set of tools has been recognized in past ITU-T standardization activities of speech algorithms. As a consequence, a library of portable, interworkable and reliable software routines has been established.

2 Software tools

To clarify the use of the set of software tools, arranged as a software tool library, the ITU-T makes the following recommendations:

1) The software tools specified in Annex A should be used as building modules of signal processing blocks to be used in the process of generation of ITU-T Recommendations, particularly those concerned with speech and audio coding algorithms.

2) Some of the tools shall be used in procedures for the verification of interoperability of ITU-T standards, mainly of speech and audio coding algorithms whose description is in terms of non-bitexact specifications.

3) The use of these modules should be made strictly in accordance with the technical instructions of their attached documentation, and should respect the following terms.

3 License and copyright

The modules in the ITU-T Software Tool Library (STL) are free software; they can be redistributed and/or modified under the terms of the "ITU-T software tools General Public License" of Annex B, as published by the ITU-T; this applies to any of the versions of the modules in the STL.

The STL has been carefully tested and it is believed that both the modules and the example programs on their usage conform to their description documents. Nevertheless, the ITU-T STL is provided "as is", in the hope that it will be useful, but without any warranty.

The STL is intended to help the scientific community to achieve new standards in telecommunications more efficiently, and for such must not be sold, entirely or in parts. The original developers, except where otherwise noted, retain ownership of their copyright, and allow their use under the terms and conditions of the "ITU-T software tools General Public License".

Annex A

List of software tools available

(This annex forms an integral part of this Recommendation)

This annex contains a list with a short description of the software tools available in the ITU-T Software Tool Library. This is referred to in the associated documentation as the Software Tool Library release 2009, or STL2009. All the routines in the modules are written in C.

a) *Example programs available*

Associated header file: ugstdemo.h

The following programs are examples of the use of the modules:

`g711demo.c`	on the use of the ITU-T G.711 module.
`g726demo.c`	on the use of the ITU-T G.726 module.
`g727demo.c`	on the use of the ITU-T G.727 module
`g722demo.c`	on the use of the ITU-T G.722 module.
`g728enc.c`	on the use of the ITU-T G.728 floating-point encoder.
`g728dec.c`	on the use of the ITU-T G.728 floating-point decoder.
`g728fpenc.c`	on the use of the ITU-T G.728 fixed-point encoder.
`g728fpdec.c`	on the use of the ITU-T G.728 fixed-point decoder.
`rpedemo.c`	on the use of the full-rate GSM 06.10 speech codec module.
`sv56demo.c`	on the use of the speech voltmeter module, and also the gain/loss routine.
`eiddemo.c`	on the use of the error insertion device for bit error insertion and frame erasure.
`eid-ev.c`	on the use of the error insertion device for bit error insertion for layered bitstreams, which can be used to apply errors to individual layers in layered bitstreams such as ITU-T G.718 or ITU-T G.729.1.
`gen-patt.c`	on the use of generating bit error pattern files for error insertion in ITU-T G.192-compliant serial bitstreams encoded files.
`gen_rate_profile.c`	on the use of fast switching rate profile generation tool.
`firdemo.c`	on the use of the FIR (finite impulse response) high-quality low-pass and band-pass filters and of the FIR-IRS filters, associated with the rate change module.
`pcmdemo.c`	on the use of the ITU-T G.712 (standard PCM) IIR (infinite impulse response) filters, associated with the rate change module.
`filter.c`	on the use of both the IIR and the FIR filters available in the rate change module.
`mnrudemo.c`	on the use of the narrow-band and wideband modulated noise reference unity (ITU-T P.81) module.

`spdemo.c`	on the use of the serialization and parallelization routines of the utility module.
`g711iplc.c`	on the use of Appendix I/G.711 Packet Loss Concealment module.
`reverb.c`	on the use of the reverberation module.
`truncate.c`	on the use of the bitstream truncation module.
`freqresp.c`	on the use of the frequency response computation tool.
`stereoop.c`	on the use of stereo file operations.

NOTE – The module for the Basic Operators does not have a demo program but it is supplemented by two tools: one to evaluate program ROM complexity for fixed-point code (`basop_cnt.c`), and another to evaluate complexity (including program ROM) of floating-point implementations (`flc_example.c`). Both reside in the Basic Operators module.

b) *Rate change module with FIR (finite impulse response) routines*

Name: firflt.c

Associated header file: firflt.h

Functions included:

`delta_sm_16khz_init`	initialize 16 kHz 1:1 ÄSM weighting filter.
`hq_down_2_to_1_init`	initialize 2:1 low-pass down-sampling filter.
`hq_down_3_to_1_init`	initialize 3:1 low-pass down-sampling filter.
`hq_up_1_to_2_init`	initialize 1:2 low-pass up-sampling filter.
`hq_up_1_to_3_init`	initialize 1:3 low-pass up-sampling filter.
`irs_8khz_init`	initialize 8-kHz P.48 IRS weighting filter.
`irs_16khz_init`	initialize 16-kHz P.48 IRS weighting filter.
`linear_phase_pb_2_to_1_init`	initialize 2:1 bandpass down-sampling filter.
`linear_phase_pb_1_to_2_init`	initialize 1:2 bandpass up-sampling filter.
`linear_phase_pb_1_to_1_init`	initialize 1:1 bandpass filter.
`mod_irs_16khz_init`	initialize 16-kHz send-side modified IRS weighting filter.
`mod_irs_48khz_init`	initialize 48-kHz send-side modified IRS weighting filter.
`psophometric_8khz_init`	initialize 1:1 ITU-T O.41 psophometric weighting filter.
`p341_16khz_init`	initialize 1:1 ITU-T P.341 send-part weighting filter for data sampled at 16 kHz.
`rx_mod_irs_16khz_init`	initialize 16-kHz modified IRS receive-side weighting filter.
`rx_mod_irs_8khz_init`	initialize 8-kHz modified IRS receive-side weighting filter.
`tia_irs_8khz_init`	initialize 8-kHz IRS weighting filter using the TIA coefficients.
`ht_irs_16khz_init`	initialize 16-kHz IRS weighting filter with a half-tilt inclination within the ITU-T P.48 mask.
`msin_16khz_init`	initialize mobile station weighting filter.
`bp5k_16khz_init`	initialize 50-Hz to 5-kHz-bandpass filter (16 kHz sampling).

`bp100_5k_16khz_init`	initialize a 100-Hz to 5-kHz-bandpass filter (16-kHz sampling).
`bp14k_32khz_init`	initialize a 50-Hz to 14-kHz-bandpass filter (32-kHz sampling).
`bp20k_48khz_init`	initialize a 20-Hz to 20-kHz-bandpass filter (48-kHz sampling).
`LP1p5_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 1.5 kHz (48-kHz sampling).
`LP35_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 3.5 kHz (48-kHz sampling).
`LP7_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 7 kHz (48-kHz sampling).
`LP10_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 10 kHz (48-kHz sampling).
`LP12_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 12 kHz at (48-kHz sampling).
`LP14_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 14 kHz at 48-kHz sampling).
`LP20_48kHz_init`	initialize a low-pass filter with a cut-off frequency of 20 kHz (48-kHz sampling).
`hq_kernel`	FIR filtering function.
`hq_reset`	clear state variables.
`hq_free`	deallocate FIR-filter memory.

c) *Rate change module with IIR routines*

Name: iirflt.c

Associated header file: iirflt.h

Functions included:

`stdpcm_kernel`	parallel-form IIR kernel filtering routine.
`stdpcm_16khz_init`	initialization of a parallel-form IIR standard PCM-filter for input and output data at 16 kHz.
`stdpcm_1_to_2_init`	as "stdpcm_16khz_init( )", but needs input with sampling frequency of 8 kHz and returns data at 16 kHz.
`stdpcm_2_to_1_init`	as "stdpcm_16khz_init( )", but needs input with sampling frequency of 16 kHz and returns data at 8 kHz.
`stdpcm_reset`	clear state variables (needed only if another signal should be processed with the same filter) for a parallel-form structure.
`stdpcm_free`	deallocate filter memory for a parallel-form state variable structure.
`cascade_iir_kernel`	cascade-form IIR filtering routine.
`iir_G712_8khz init`	initialization of a cascade-form IIR standard PCM filter for data sampled at 8 kHz.

`iir_irs_8khz_init`	initialization of a cascade-form IIR P.48 IRS filter for data sampled at 8 kHz.
`iir_casc_lp_3_to_1_init`	initialization of a cascade-form IIR low-pass filter for asynchronization filtering of data and downsampling by a factor of 3:1.
`iir_casc_lp_1_to_3_init`	initialization of a cascade-form IIR low-pass filter for asynchronization filtering of data and upsampling by a factor of 3:1.
`cascade_iir_reset`	clear state variables (needed only if another signal should be processed with the same filter) for a cascade-form structure.
`cascade_iir_free`	deallocate filter memory for a cascade-form state variable structure.
`direct_iir_kernel`	direct-form IIR filtering routine.
`iir_dir_dc_removal_init`	Initialize a direct-form IIR filter structure for a 1:1 DC removal filtering.
`direct_reset`	clear state variables (needed only if another signal should be processed with the same filter) for a direct-form structure.
`direct_iir_free`	deallocate filter memory for a direct-form state variable structure.

d) *Error insertion module*

Name: eid.c

Associated header file: eid.h

Functions included:

`open_eid`	initializes the error pattern generator (for single-bit errors, burst bit-errors, or single frame erasures).
`open_burst_eid`	initializes the burst frame erasure pattern generator.
`reset_burst_eid`	reinitializes the burst frame erasure pattern generator.
`BER_generator`	generates a bit error sequence with properties defined by "open_eid".
`FER_generator_random`	generates a random frame erasure sequence with properties, defined by "open_eid".
`FER_generator_burst`	generates a burst frame erasure sequence with properties, defined by "open_burst_eid".
`BER_insertion`	modifies the input data bits according to the error pattern, stored in a buffer.
`FER_module`	frame erasure module.
`close_eid`	frees memory allocated to the EID state variable buffer.

e) *G.711 module*

Name: g711.c

Associated header file: g711.h

Functions included:

`alaw_compress`	compands 1 vector of linear PCM samples to A-law; uses 13 Most Significant Bits (MSBs) from input and 8 Least Significant Bits (LSBs) on output.
`alaw_expand`	expands 1 vector of A-law samples to linear PCM; uses 8 LSBs from input and 13 MSBs on output.
`ulaw_compress`	compands 1 vector of linear PCM samples to μ-law; uses 14 MSBs from input and 8 LSBs on output.
`ulaw_expand`	expands 1 vector of μ-law samples to linear PCM; uses 8 LSBs from input and 14 MSBs on output.

f) *Appendix I/G.711 Packet Loss Concealment module*

Name: lowcfe.c

Associated header file: lowcfe.h

Functions included:

`g711plc_construct`	LowcFE Constructor.
`g711plc_dofe`	generate the synthetic signal.
`g711plc_addtohistory`	a good frame was received and decoded, add the frame to history buffer.

g) *G.726 module*

Name: g726.c

Associated header file: g726.h

Functions included:

`G726_encode`	ITU-T G.726 encoder at 40, 32, 24 and 16 kbit/s.
`G726_decode`	ITU-T G.726 decoder at 40, 32, 24 and 16 kbit/s.

h) *Modulated noise reference unit module*

Name: mnru.c

Associated header file: mnru.h

Functions included:

`MNRU_process`	module for addition of modulated noise to a vector of samples, according to Rec. ITU-T P.810, for both the narrow-band and the wideband models.

i) *Speech voltmeter module*

Name: sv-p56.c

Associated header file: sv-p56.h

Functions included:

`init_speech_voltmeter`	initializes a speech voltmeter state variable.
`speech_voltmeter`	measurement of the active speech level of data in a buffer according to Rec. ITU-T P.56.

j) *Module with UGST utilities*

Name: ugst-utl.c

Associated header file: ugst-utl.h

Functions included:

`scale`	gain/loss insertion algorithm.
`sh2fl_16bit`	conversion of two's complement, 16-bit integer to floating point.
`sh2fl_15bit`	conversion of two's complement, 15-bit integer to floating point.
`sh2fl_14bit`	conversion of two's complement, 14-bit integer to floating point.
`sh2fl_13bit`	conversion of two's complement, 13-bit integer to floating point.
`sh2fl_12bit`	conversion of two's complement, 12-bit integer to floating point.
`sh2fl`	generic function for conversion from integer to floating point.
`sh2fl_alt`	alternate (faster) implementation of sh2fl, with compulsory range conversion.
`fl2sh_16bit`	conversion of floating point data to two's complement, 16-bit integer.
`fl2sh_15bit`	conversion of floating point data to two's complement, 15-bit integer.
`fl2sh_14bit`	conversion of floating point data to two's complement, 14-bit integer.
`fl2sh_13bit`	conversion of floating point data to two's complement, 13-bit integer.
`fl2sh_12bit`	conversion of floating point data to two's complement, 12-bit integer.
`fl2sh`	generic function for conversion from floating point to integer.
`serialize_left_justified`	serialization for left-justified data.
`serialize_right_justified`	serialization for right-justified data.
`parallelize_left_justified`	parallelization for left-justified data.
`parallelize_right_justified`	parallelization for right-justified data.

k) *G.722 module*

Name: g722.c

Associated header file: g722.h

Functions included:

`G722_encode`	ITU-T G.722 wideband speech encoder at 64 kbit/s.
`G722_decode`	ITU-T G.722 wideband speech decoder at 64, 56 and 48 kbit/s.
`g722_reset_encoder`	initialization of the ITU-T G.722 encoder state variable.
`g722_reset_decoder`	initialization of the ITU-T G.722 decoder state variable.

l) *RPE-LTP module*

Name: rpeltp.c

Associated header file: rpeltp.h

Functions included:

`rpeltp_encode`	GSM 06.10 full-rate RPE-LTP speech encoder at 13 kbit/s.
`rpeltp_decode`	GSM 06.10 full-rate RPE-LTP speech decoder at 13 kbit/s.
`rpeltp_init`	initialize memory for the RPE-LTP state variables.
`rpeltp_delete`	release memory previously allocated for the RPE-LTP state variables.

m) *G.727 module*

Name: g727.c

Associated header file: g727.h

Functions included:

`G727_encode`	ITU-T G.727 encoder at 40, 32, 24 and 16 kbit/s.
`G727_decode`	ITU-T G.727 decoder at 40, 32, 24 and 16 kbit/s.

n) *Basic Operators*

Name: basop32.c, enh1632.c, enh40.c

Associated header file: stl.h

Variable definitions:

- v1, v2: 16-bit variables
- L_v1, L_v2, L_v3: 32-bit variables
- L40_v1, L40_v2, L40_v3: 40-bit variables

Functions included:

`add(v1, v2)`	Performs the addition (v1+v2) with overflow control and saturation; the 16-bit result is set at +32767 when overflow occurs or at –32768 when underflow occurs.
`sub(v1, v2)`	Performs the subtraction (v1–v2) with overflow control and saturation; the 16-bit result is set at +32767 when overflow occurs or at –32768 when underflow occurs.
`abs_s(v1)`	Absolute value of v1. If v1 is –32768, returns 32767.

`shl(v1, v2)`	Arithmetically shifts the 16-bit input v1 left by v2 positions. Zero fills the v2 LSB of the result. If v2 is negative, arithmetically shifts v1 right by –v2 with sign extension. Saturates the result in case of underflows or overflows.
`shr(v1, v2)`	Arithmetically shifts the 16-bit input v1 right v2 positions with sign extension. If v2 is negative, arithmetically shifts v1 left by –v2 and zero fills the –v2 LSB of the result: `shr(v1, v2) = shl(v1, -v2)` Saturates the result in case of underflows or overflows.
`negate(v1)`	Negates v1 with saturation, saturate in the case when input is –32768: `negate(v1) = sub(0, v1)`
`s_max(v1, v2)`	Compares two 16-bit variables v1 and v2 and returns the maximum value.
`s_min(v1, v2)`	Compares two 16-bit variables v1 and v2 and returns the minimum value.
`norm_s(v1)`	Produces the number of left shifts needed to normalize the 16-bit variable v1 for positive values on the interval with minimum of 16384 and maximum 32767, and for negative values on the interval with minimum of –32768 and maximum of –16384; in order to normalize the result, the following operation must be done: `norm_v1 = shl(v1, norm_s(v1))`
`L_add(L_v1, L_v2)`	32-bit addition of the two 32-bit variables (L_v1+L_v2) with overflow control and saturation; the result is set at +2147483647 when overflow occurs or at –2147483648 when underflow occurs.
`L_sub(L_v1, L_v2)`	32-bit subtraction of the two 32-bit variables (L_v1–L_v2) with overflow control and saturation; the result is set at +2147483647 when overflow occurs or at –2147483648 when underflow occurs.
`L_abs(L_v1)`	Absolute value of L_v1, with L_abs(–2147483648) = 2147483647.
`L_shl(L_v1, v2)`	Arithmetically shifts the 32-bit input L_v1 left v2 positions. Zero fills the v2 LSB of the result. If v2 is negative, arithmetically shifts L_v1 right by –v2 with sign extension. Saturates the result in case of underflows or overflows.
`L_shr(L_v1, v2)`	Arithmetically shifts the 32-bit input L_v1 right v2 positions with sign extension. If v2 is negative, arithmetically shifts L_v1 left by –v2 and zero fills the –v2 LSB of the result. Saturates the result in case of underflows or overflows.
`L_negate(L_v1)`	Negates the 32-bit L_v1 with saturation, saturate in the case where input is –2147483648.
`L_max(L_v1, L_v2)`	Compares two 32-bit variables L_v1 and L_v2 and returns the maximum value.

`L_min(L_v1, L_v2)`	Compares two 32-bit variables L_v1 and L_v2 and returns the minimum value.
`norm_l(L_v1)`	Produces the number of left shifts needed to normalize the 32-bit variable L_v1 for positive values on the interval with minimum of 1073741824 and maximum 2147483647, and for negative values on the interval with minimum of –2147483648 and maximum of –1073741824; in order to normalize the result, the following operation must be done: `L_norm_v1 = L_shl( L_v1, norm_l( L_v1))`
`L_mult(v1, v2)`	L_mult implements the 32-bit result of the multiplication of v1 times v2 with one shift left, i.e., `L_mult(v1, v2) = L_shl( (v1 * v2), 1)` Note that L_mult(–32768,–32768) = 2147483647.
`L_mult0(v1, v2)`	L_mult0 implements the 32-bit result of the multiplication of v1 times v2 without left shift, i.e., `L_mult(v1, v2) = (v1 * v2)`
`mult(v1, v2)`	Performs the multiplication of v1 by v2 and gives a 16-bit result which is scaled, i.e., `mult(v1, v2) = extract_l( L_shr( (v1 times v2), 15))` Note that mult(–32768,–32768) = 32767.

`mult_r(v1, v2)`	Same as mult() but with rounding, i.e.,
	`mult_r( v1, v2) = extract_l( L_shr( ((v1 * v2) + 16384), 15))`
	`and mult_r(-32768, -32768) = 32767.`

`L_mac(L_v3, v1, v2)`	Multiplies v1 by v2 and shifts the result left by 1. Adds the 32-bit result to L_v3 with saturation, returns a 32-bit result: `L_mac( L_v3, v1, v2) = L_add(L_v3, L_mult( v1, v2))`
`L_mac0(L_v3, v1, v2)`	Multiplies v1 by v2 without left shift. Adds the 32-bit result to L_v3 with saturation, returning a 32-bit result: `L_mac(L_v3, v1, v2) = L_add(vL_v3, L_mult0(vv1, v2))`
`L_macNs(L_v3, v1, v2)`	Multiplies v1 by v2 and shifts the result left by 1. Adds the 32-bit result to L_v3 without saturation, returns a 32-bit result. Generates carry and overflow values: `L_macNs( L_v3, v1, v2) = L_add_c( L_v3, L_mult(v1, v2))`

`mac_r(L_v3, v1, v2)`

Multiplies v1 by v2 and shifts the result left by 1. Adds the 32-bit result to L_v3 with saturation. Rounds the 16 least significant bits of the result into the 16 most significant bits with saturation and shifts the result right by 16. Returns a 16-bit result.

```
mac_r( L_v3, v1, v2) = round( L_mac( L_v3, v1, v2)) =
extract_h( L_add( L_add( L_v3, L_mult( v1, v2)), 32768))
```

`L_msu(L_v3, v1, v2)`

Multiplies v1 by v2 and shifts the result left by 1. Subtracts the 32-bit result from L_v3 with saturation, returns a 32-bit result:

```
L_msu(L_v3, v1, v2) = L_sub(L_v3, L_mult(v1, v2))
```

`L_msu0(L_v3, v1, v2)`

Multiplies v1 by v2 without left shift. Subtracts the 32-bit result from L_v3 with saturation, returning a 32-bit result:

```
L_msu(L_v3, v1, v2) = L_sub(L_v3, L_mult0( v1, v2))
```

`L_msuNs(L_v3, v1, v2)`

Multiplies v1 by v2 and shifts the result left by 1. Subtracts the 32-bit result from L_v3 without saturation, returns a 32-bit result. Generates carry and overflow values:

```
L_msuNs( L_v3, v1, v2) = L_sub_c(L_v3, L_mult( v1, v2))
```

`msu_r(L_v3, v1, v2)`

Multiplies v1 by v2 and shifts the result left by 1. Subtracts the 32-bit result from L_v3 with saturation. Rounds the 16 least significant bits of the result into the 16 bits with saturation and shifts the result right by 16. Returns a 16-bit result.

```
msu_r( L_v3, v1, v2) = round( L_msu( L_v3, v1, v2)) =
extract_h( L_add( L_sub( L_v3, L_mult( v1, v2)), 32768))
```

`s_and(v1, v2)`

Performs a bit wise AND between the two 16-bit variables v1 and v2.

`s_or(v1, v2)`

Performs a bit wise OR between the two 16-bit variables v1 and v2.

`s_xor(v1, v2)`

Performs a bit wise XOR between the two 16-bit variables v1 and v2.

`lshl(v1, v2)`

Logically shifts left the 16-bit variable v1 by v2 positions:

if v2 is negative, v1 is shifted to the least significant bits by (–v2) positions with insertion of 0 at the most significant bit.

if v2 is positive, v1 is shifted to the most significant bits by (v2) positions without saturation control.

`lshr(v1, v2)` Logically shifts right the 16-bit variable v1 by v2 positions:

if v2 is positive, v1 is shifted to the least significant bits by (v2) positions with insertion of 0 at the most significant bit.

if v2 is negative, v1 is shifted to the most significant bits by (–v2) positions without saturation control.

`L_and(L_v1, L_v2)` Performs a bit wise AND between the two 32-bit variables L_v1 and L_v2.

`L_or(L_v1, L_v2)` Performs a bit wise OR between the two 32-bit variables L_v1 and L_v2.

`L_xor(L_v1, L_v2)` Performs a bit wise XOR between the two 32-bit variables L_v1 and L_v2.

`L_lshl(L_v1, v2)` Logically shifts left the 32-bit variable L_v1 by v2 positions:

if v2 is negative, L_v1 is shifted to the least significant bits by (–v2) positions with insertion of 0 at the most significant bit.

if v2 is positive, L_v1 is shifted to the most significant bits by (v2) positions without saturation control.

`L_lshr(L_v1, v2)` Logically shifts right the 32-bit variable L_v1 by v2 positions:

if v2 is positive, L_v1 is shifted to the least significant bits by (v2) positions with insertion of 0 at the most significant bit.

if v2 is negative, L_v1 is shifted to the most significant bits by (–v2) positions without saturation control.

`extract_h(L_v1)` Returns the 16 most significant bits of L_v1.

`extract_l(L_v1)` Returns the 16 least significant bits of L_v1.

`round(L_v1)` Rounds the lower 16 bits of the 32-bit input number into the most significant 16 bits with saturation. Shifts the resulting bits right by 16 and returns the 16-bit number:

```
round(L_v1) = extract_h( L_add( L_v1, 32768))
```

`L_deposit_h(v1)` Deposits the 16-bit v1 into the 16-bit most significant bits of the 32-bit output. The 16 least significant bits of the output are zeroed.

`L_deposit_l(v1)` Deposits the 16-bit v1 into the 16-bit least significant bits of the 32-bit output. The 16 most significant bits of the output are sign extended.

`L_add_c(L_v1, L_v2)` Performs the 32-bit addition with carry. No saturation. Generates carry and overflow values. The carry and overflow values are binary variables which can be tested and assigned values.

`L_sub_c(L_v1, L_v2)` Performs the 32-bit subtraction with carry (borrow). Generates carry (borrow) and overflow values. No saturation. The carry and overflow values are binary variables which can be tested and assigned values.

`shr_r(v1, v2)`	Same as shr(v1, v2) but with rounding. Saturates the result in case of underflows or overflows.
	if v2 is strictly greater than zero, then
	`if (sub( shl ( shr( v1,v2), 1), shr( v1, sub(v2, 1))) == 0)`
	`then shr_r(v1, v2) = shr(v1, v2)`
	`else shr_r(v1, v2) = add(shr(v1, v2), 1)`
	On the other hand, if v2 is lower than or equal to zero, then
	`shr_r(v1, v2) = shr(v1, v2)`

`shl_r(v1, v2)`	Same as shl(v1, v2) but with rounding. Saturates the result in case of underflows or overflows: `shl_r( v1, v2) = shr_r( v1, -v2)` In the previous version of the STL-basic operators, this operator is called shift_r(v1, v2); both names can be used.

`L_shr_r(L_v1, v2)`	Same as L_shr(v1, v2) but with rounding. Saturates the result in case of underflows or overflows:
	if v2 is strictly greater than zero, then
	`if( L_sub( L_shl( L_shr( L_v1, v2), 1), L_shr( L_v1, sub( v2, 1)))) == 0`
	`then L_shr_r( L_v1, v2) = L_shr( L_v1, v2)`
	`else L_shr_r( L_v1, v2) = L_add( L_shr( L_v1, v2), 1)`
	On the other hand, if v2 is less than or equal to zero, then
	`L_shr_r( L_v1, v2) = L_shr( L_v1, v2)`

`L_shl_r(L_v1, v2)`	Same as L_shl(L_v1, v2) but with rounding. Saturates the result in case of underflows or overflows. `L_shift_r( L_v1, v2) = L_shr_r( L_v1, -v2)` In the previous version of the STL-basic operators, this operator is called L_shift_r(L_v1, v2); both names can be used.
`i_mult(v1, v2)`	Multiplies two 16-bit variables v1 and v2 returning a 16-bit value with overflow control.
`rotl(v1, v2, *v3)`	Rotates the 16-bit variable v1 by 1 bit to the most significant bits. Bit 0 of v2 is copied to the least significant bit of the result before it is returned. The most significant bit of v1 is copied to the bit 0 of v3 variable.
`rotr(v1, v2, *v3)`	Rotates the 16-bit variable v1 by 1 bit to the least significant bits. Bit 0 of v2 is copied to the most significant bit of the result before it is returned. The least significant bit of v1 is copied to the bit 0 of v3 variable.
`L_rotl(L_v1, v2, *v3)`	Rotates the 32-bit variable L_v1 by 1 bit to the most significant bits. Bit 0 of v2 is copied to the least significant bit of the result before it is returned. The most significant bit of L_v1 is copied to the bit 0 of v3 variable.

`L_rotr(L_v1, v2, *v3)`	Rotates the 32-bit variable L_v1 by 1 bit to the least significant bits. Bit 0 of v2 is copied to the most significant bit of the result before it is returned. The least significant bit of L_v1 is copied to the bit 0 of v3 variable.
`L_sat(L_v1)`	Long (32-bit) L_v1 is set to 2147483647 if an overflow occurred, or –2147483648 if an underflow occurred, on the most recent L_add_c(), L_sub_c(), L_macNs() or L_msuNs() operations. The carry and overflow values are binary variables which can be tested and assigned values.
`L_mls(L_v1, v2)`	Performs a multiplication of a 32-bit variable L_v1 by a 16-bit variable v2, returning a 32-bit value.
`div_s(v1, v2)`	Produces a result which is the fractional integer division of v1 by v2. Values in v1 and v2 must be positive and v2 must be greater than or equal to v1. The result is positive (leading bit equal to 0) and truncated to 16 bits. If v1 equals v2, then div(v1, v2) = 32767.
`div_l(L_v1, v2)`	Produces a result which is the fractional integer division of a positive 32-bit variable L_v1 by a positive 16-bit variable v2. The result is positive (leading bit equal to 0) and truncated to 16 bits.
`L40_add(L40_v1, L40_v2)`	Adds the two 40-bit variables L40_v1 and L40_v2 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.
`L40_sub(L40_v1, L40_v2)`	Subtracts the two 40-bit variables L40_v2 from L40_v1 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.
`L40_abs(L40_v1)`	Returns the absolute value of the 40-bit variable L40_v1 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.
`L40_shl(L40_v1, v2)`	Arithmetically shifts left the 40-bit variable L40_v1 by v2 positions: if v2 is negative, L40_v1 is shifted to the least significant bits by (–v2) positions with extension of the sign bit. if v2 is positive, L40_v1 is shifted to the most significant bits by (v2) positions without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.
`L40_shr(L40_v1, v2)`	Arithmetically shifts right the 40-bit variable L40_v1 by v2 positions: if v2 is positive, L40_v1 is shifted to the least significant bits by (v2) positions with extension of the sign bit. if v2 is negative, L40_v1 is shifted to the most significant bits by (–v2) positions without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.
`L40_negate(L40_v1)`	Negates the 40-bit variable L40_v1 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.

`L40_max(L40_v1, L40_v2)`	Compares two 40-bit variables L40_v1 and L40_v2 and returns the maximum value.
`L40_min(L40_v1, L40_v2)`	Compares two 40-bit variables L40_v1 and L40_v2 and returns the minimum value.
`norm_L40(L40_v1)`	Produces the number of left shifts needed to normalize the 40-bit variable L40_v1 for positive values on the interval with minimum of 1073741824 and maximum 2147483647, and for negative values on the interval with minimum of –2147483648 and maximum of –1073741824; in order to normalize the result, the following operation must be done: `L40_norm_v1 = L40_shl( L40_v1, norm_L40( L40_v1))`
`L40_mult(v1, v2)`	Multiplies the 2 signed 16-bit variables v1 and v2 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution. The operation is performed in fractional mode: v1 and v2 are supposed to be in 1Q15 format. The result is produced in 9Q31 format.
`L40_mac(L40_v1, v2, v3)`	Equivalent to: `L40_add( L40_v1, L40_mult( v2, v3))`
`L40_msu(L40_v1, v2, v3)`	Equivalent to: `L40_sub( L40_v1, L40_mult( v2, v3))`
`L40_lshl(L40_v1, v2)`	Logically shifts left the 40-bit variable L40_v1 by v2 positions: if v2 is negative, L40_v1 is shifted to the least significant bits by (–v2) positions with insertion of 0 at the most significant bit. if v2 is positive, L40_v1 is shifted to the most significant bits by (v2) positions without saturation control.
`L40_lshr(L40_v1, v2)`	Logically shifts right the 40-bit variable L40_v1 by v2 positions: if v2 is positive, L40_v1 is shifted to the least significant bits by (v2) positions with insertion of 0 at the most significant bit. if v2 is negative, L40_v1 is shifted to the most significant bits by (–v2) positions without saturation control.
`Extract40_H(L40_v1)`	Returns the bits [31..16] of L40_v1.
`Extract40_L(L40_v1)`	Returns the bits [15..00] of L40_v1.
`round40(L40_v1)`	Equivalent to: `extract_h( L_saturate40( L40_round( L40_v1)))`
`L_Extract40(L40_v1)`	Returns the bits [31..00] of L40_v1.

`L_saturate40(L40_v1)`	If L40_v1 is greater than 2147483647, returns 2147483647. If L40_v1 is lower than –2147483648, returns –2147483648. If not, equivalent to: `L_Extract40( L40_v1)`
`L40_deposit_h(v1)`	Deposits the 16-bit variable v1 in the bits [31..16] of the return value: the return value bits [15..0] are set to 0 and the bits [39..32] sign extend v1 sign bit.
`L40_deposit_l(v1)`	Deposits the 16-bit variable v1 in the bits [15..0] of the return value: the return value bits [39..16] sign extend v1 sign bit.
`L40_deposit32(L_v1)`	Deposits the 32-bit variable L_v1 in the bits [31..0] of the return value: the return value bits [39..32] sign extend L_v1 sign bit.
`L40_round(L40_v1)`	Performs a rounding to the infinite on the 40-bit variable L40_v1. 32768 is added to L40_v1 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution. The end-result 16 LSB are cleared to 0.
`mac_r40(L40_v1, v2, v3)`	Equivalent to: `round40( L40_mac( L40_v1, v2, v3))`
`msu_r40(L40_v1, v2, v3)`	Equivalent to: `round40( L40_msu( L40_v1, v2, v3))`
`Mpy_32_16_ss(L_v1, v2, *L_v3_h, *v3_l)`	Multiplies the 2 signed values L_v1 (32-bit) and v2 (16-bit) with saturation control on 48 bits. The operation is performed in fractional mode: When L_v1 is in 1Q31 format, and v2 is in 1Q15 format, the result is produced in 1Q47 format: L_v3_h bears the 32 most significant bits while v3_l bears the 16 least significant bits.
`L40_shr_r(L40_v1, v2)`	Arithmetically shifts the 40-bit variable L40_v1 by v2 positions to the least significant bits and rounds the result. It is equivalent to L40_shr(L40_v1, v2) except that if v2 is positive and the last shifted out bit is 1, then the shifted result is incremented by 1 without saturation control on 40 bits. Any detected overflow on 40 bits will exit execution.
`L40_shl_r(L40_v1, v2)`	Arithmetically shifts the 40-bit variable L40_v1 by v2 positions to the most significant bits and rounds the result. It is equivalent to L40_shl(L40_var1, v2) except if v2 is negative. In this case, it does the same as L40_shr_r(L40_v1, (–v2)).

`L40_set(L40_v1)`	Assigns a 40-bit constant to the returned 40-bit variable.
`Mpy_32_32_ss(L_v1, L_v2, *L_v3_h, *L_v3_l)`	Multiplies the 2 signed 32-bit values L_v1 and L_v2 with saturation control on 64 bits. The operation is performed in fractional mode: When L_v1 and L_v2 are in 1Q31 format, the result is produced in 1Q63 format: L_v3_h bears the 32 most significant bits while L_v3_l bears the 32 least significant bits.

The Basic Operators module is supplemented by two tools: one to evaluate program ROM complexity for fixed-point code, and another to evaluate complexity (including program ROM) of floating-point implementations.

n.1) *Program ROM estimation tool for fixed-point C code*

Name: basop_cnt.c

Associated header file: None.

Usage: `basop cnt input.c [result_file_name.txt]`

The basop_cnt tool estimates the program ROM of applications written using the ITU-T Basic Operator libraries. It counts the number of calls to basic operators in the input C source file, and also the number of calls to user-defined functions. The sum of these two numbers gives an estimation of the required PROM.

n.2) *Complexity evaluation tool for floating-point C code*

Name: flc.c

Associated header file: flc.h

Functions included:

`FLC_init`	Initialize the floating-point counters.
`FLC_sub_start`	Marks the start of a subroutine/subsection.
`FLC_sub_end`	Marks the end of a subroutine/subsection.
`FLC_end`	Computes and prints complexity, i.e., floating-point counter results.
`FLC_frame_update`	Marks the end of a frame processing to keep track of the per-frame maxima.

o) *Reverberation module*

Name: reverb-lib.c

Associated header file: reverb-lib.h

Functions included:

`conv`	Convolution routine.
`shift`	Shift elements of a vector for the block-based convolution.

p) *Bit stream truncation module*

Name: trunc-lib.c

Associated header file: trunc-lib.h

Functions included:

`trunc`	Frame truncation routine.

q) *Frequency response calculation module*

Name: fft.c

Associated header file: fft.h

Functions included:

`rdft`	Discrete Fourier Transform for real signals.
`genHanning`	Hanning window generation routine.
`powSpect`	Power spectrum computation routine.

Annex B

ITU-T software tools General Public License

(This annex forms an integral part of this Recommendation)

Terms and conditions

B.1 This License Agreement applies to any module or other work related to the ITU-T Software Tool Library, and developed by the User's Group on Software Tools. The term "Module", below, refers to any such module or work, and a "work based on the Module" means either the Module or any work containing the Module or a portion of it, either verbatim or with modifications. Each licensee is addressed as "you".

B.2 You may copy and distribute verbatim copies of the Module's source code as you receive it, in any medium, provided that you:

– conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty;

– keep intact all the notices that refer to this General Public License and to the absence of any warranty; and

– give any other recipients of the Module a copy of this General Public License along with the Module.

You may charge a fee for the physical act of transferring a copy.

B.3 You may modify your copy or copies of the Module or any portion of it, and copy and distribute such modifications under the terms of B.1, provided that you also do the following:

- cause the modified files to carry prominent notices stating that you changed the files and the date of any change; and
- cause the whole of any work that you distribute or publish, that in whole or in part contains the Module or any part thereof, either with or without modifications, to be licensed at no charge to all third parties under the terms of this General Public License (except that you may choose to grant warranty protection to some or all third parties, at your option);
- if the modified module normally reads commands interactively when run, you must cause it, when started running for such interactive use in the simplest and most usual way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the module under these conditions, and telling the user how to view a copy of this General Public License.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Mere aggregation of another independent work with the Module (or its derivative) on a volume of a storage or distribution medium does not bring the other work under the scope of these terms.

B.4 You may copy and distribute the Module (or a portion or derivative of it, under B.2) in object code or executable form under the terms of B.1 and B.2, provided that you also do one of the following:

- accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of B.1 and B.2; or
- accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal charge for the cost of distribution) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of B.1 and B.2; or

- accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for non-commercial distribution and only if you received the module in object code or executable form alone.)

Source code for a work means the preferred form of the work for making modifications to it. For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are standard libraries that accompany the operating system on which the executable file runs, or for standard header files or definitions files that accompany that operating system.

B.5 You may not copy, modify, sublicense, distribute or transfer the Module except as expressly provided under this General Public License. Any attempt otherwise to copy, modify, sublicense, distribute or transfer the Module is void, and will automatically terminate your rights to use the Module under this License. However, parties who have received copies, or rights to use copies, from you under this General Public License will not have their licenses terminated so long as such parties remain in full compliance.

B.6 By copying, distributing or modifying the Module (or any work based on the Module) you indicate your acceptance of this license to do so, and all its terms and conditions.

B.7 Each time you redistribute the Module (or any work based on the Module), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Module subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

B.8 The ITU-T may publish revised and/or new versions of this General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Module specifies a version number of the license which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the ITU-T. If the Module does not specify a version number of the license, you may choose any version ever published by the ITU-T.

B.9 If you wish to incorporate parts of the Module into other free modules whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the ITU-T, write to the ITU-T Secretariat; exceptions may be made for this. This decision will be guided by the two goals of preserving the free status of all derivatives of this free software and of promoting the sharing and reuse of software generally.

B.10 Because the Module is licensed free of charge, there is no warranty for the Module, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the Module "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Module is with you. Should the Module prove defective, you assume the cost of all necessary servicing, repair or correction.

B.11 In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the Module as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the Module (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the Module to operate with any other modules), even if such holder or other party has been advised of the possibility of such damages.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	**Transmission systems and media, digital systems and networks**
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2010